COP 3330
Homework 4

Out: 2/28/12 (Tuesday)
Due: 3/16/12 (Friday) at 11:55 PM Webcourses time
Late submissions accepted until 3/18/12 (Sunday) at 11:55 PM

Objective

1. To help you become comfortable with using inheritance and interfaces to construct type hierarchies.
2. To familiarize you with programming from specifications.

Code

You may discuss high-level ideas of how to solve these problems with your classmates, but do not discuss actual source code (except for boiler-plate code such as opening a file for reading, reading user input, etc.). All source code should be yours and yours alone. **Do not:**

- Share your source code with anyone
- Ask anyone to share his/her source code with you
- Ask/pay a programming forum/community to write your homework for you
- Pair program
- Do anything else that you know is deceptive, dishonest, or misleading in any way

Problem: Adventurer Assistance, Inc.

Fresh from your successful exploits with the famed Tallahassee Smith, you've spotted a market opportunity and decided to go into business for yourself. Your startup, Adventurer Assistance, provides online guidance to teams of adventurers in sticky situations. Your first contract involves providing mission control for teams of three exploring underground cave systems suspected to have been built by aliens, dwarves, the Great Old Ones, or perhaps even all three. Their aim is to reach a treasure cache whose location they have identified. But first they need to navigate through the dark and treacherous caves, and helping them do that right is your job.

Your helpful assistants have already written a graphical user interface for your program, and provided a set of specifications for classes that you must write in order to make it work. When you're done, you'll be able to send commands to the three team members, simply by clicking on their icons and moving them around with arrows. This allows you to trick online gamers into guiding expeditions, by having them think they're playing a 2-D maze game instead.

In practical terms, you will not write a *main()* method at all, just the functionality that *main()* already expects. In effect, you're writing a library for the use of the prewritten GUI code.

The caves are laid out in a regular two-dimensional grid pattern, which you have named the **Board**. Each cell of the grid represents a single cave in the cave system. A cave is big enough to hold exactly one person (the builders were apparently rather small), and is connected to its four neighboring caves to the North, South, East and West.

However, not all caves are accessible. While some caves are **open**, others may have been **blocked** by a cave-in, and cannot be entered. Some others contain **bottomless pits** that will be the end of any unwary explorer that walks into them. And finally, certain caves are **teleport locations** that will instantly transport anyone who enters to a random open cave anywhere on the Board. These are especially problematic because we usually can't tell where they are, except by entering and getting teleported.

A team is composed of three people, with unique specialties:

1. **Miner:** A demolition expert. If he moves into a location with a cave-in, he will destroy the rocks blocking it off, converting it into an open cell.
2. **Filler**: A vertical descent safety specialist who can fill in bottomless pits. If he enters a location with a pit in it, the pit will be filled in and the cave will be considered open.
3. **Adventurer:** The intrepid leader of the expedition. She is the one who **must reach the treasure cache** in order to achieve **mission success**, because the other two don't look as good on camera, and 21st century adventuring is really all about the post-adventure merchandising deals. She is equipped with a cutting-edge Teleport Interdiction System. As a result, if she enters a teleport location, she will not get teleported. Instead, that location will be marked on the Board, so that the other two can avoid it (or risk it, if necessary). Unfortunately, she can't enter blocked-off caves, and will be killed if she walks into a cave with a pit in it. If she dies, the mission ends, and is considered a **failure**.

Neither of these three can leave the cave – the boundaries are impenetrable. The only way out is to get the Adventurer to the treasure in the bottom-right corner of the Board.

At the start of the game, a random board is generated. (The reason for this is complicated, and involves a series of unfortunate accidents involving a time machine and a random number generator.) The **Adventurer** is at the top-left location, and the **Miner** and **Filler** are at random open caves on the **Board**, having gone in earlier. Such is life for a professional sidekick.

To model this situation, you must write the following set of classes. (**See documentation** for precise technical details on what you should implement.)

1. **Cave:** A class that represents a single cave. Its instance variables contain information on its position on the Board, the type of cave it is (open, blocked, pit, teleport), whether it has been marked (for teleport caves), whether it's currently occupied by one of the Characters, and various instance methods described fully in the docs.
2. **Board:** Represents the game board. It contains a 2-D array of **Caves**, and is responsible for generating a random cave system, among other things.
3. **Character:** An **abstract** base class representing members of the team. It handles common functionality, like moving from a cave to an adjacent cave. The classes **Adventurer, Miner, and Filler** are descended from it, and override its methods to model their unique behavior. It implements the ***CaveWorker*** interface.
4. **Adventurer:** A **subclass of Character** that represents the leader of the expedition. Incorporates the special ability to be immune to teleportation and mark teleport locations.
5. **Miner:** A **subclass of Character** that represents the miner, and incorporates his special ability to clear blocked caves.
6. **Filler:** A **subclass of Character** that represents the filler, and incorporates his special ability to fill up pits.
7. ***CaveWorker*:** An interface that represents the ability to change a non-open location to an open one (by filling a pit or blowing up rocks).

You will be provided with the **AdventureGame** class that will generate the game interface and handle input/output. You **SHOULD NOT** need to alter this file.

Input/Output

You don't need to worry about input/output—this will be handled by the prewritten class we will provide.

Sample I/O

Watch the video for an example of a working game. Take note of all the necessary functionality, and make sure your program implements all of it!

http://youtu.be/RaY2Nfo-7j0

Extra Credit (Optional)

An additional Webcourses assignment will be created, marked Assignment 4E. This is the extra credit assignment. On this one, you can add new features, overhaul the GUI, soup up the game in crazy ways – basically be as creative as you want, possibly wasting your entire Spring Break. Up to 20 points of extra credit are available for this, which comes to 7 extra points on your final grade at the end of class. We'll show off the coolest submissions in class.

NOTE: You still need to submit the original assignment! Please avoid adding extra features to the original one – reserve your creativity for the extra credit.
Some ideas for the extra credit are:

1) Keep track of how many moves a game took, and allow the player to 'start over' after winning to try and beat a level in fewer moves instead of just starting a new random game every time.
2) Keep statistics and make them viewable.
3) Allow the user to undo a bad move (like going back in time).
4) Show some celebration when the user wins, like fireworks on the screen.
5) Use path finding to detect unwinnable games.

Deliverables

Submit the source files over Webcourses as an attachment. You need to submit the 7 files (Cave.java, Board.java, Character.java, Adventurer.java, Miner.java, Filler.java, and CaveWorker.java). **In particular, *do not* submit any .class files!!! This will result in 0 credit for the assignment.** You must send your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will **not** be accepted.

For the bonus assignment, submit all java files you need for your code to work (including the 7 from before, any new classes you create for the assignment, and the modified AdventureGame.java).

Restrictions

Your program must compile using Java 6.0 or later on the command prompt. It's okay to develop your program using the IDE of your choice, but the TAs will use the command prompt to compile your code and run the programs. Your code should include a header comment with the following information:

Your name
Course number, section number
Description of the code

You **WILL** lose credit if this information is not found in the beginning of each of your programs. You should also include **inline comments** to describe different parts of your program where appropriate.

Execution and Grading Instructions

1. Download the source *.java* files and place in a folder.
2. Check the source code of each program to make sure it contains header comments, inline comments and reasonable use of variable names. Check to make sure all required methods from the documentation exists and all names are consistent with naming in the provided API.
3. Run the game and test to see if all the required functionality is present.
4. If all required functionality is present and works correctly, give full credit for execution. Otherwise, give partial credit based on the situation.